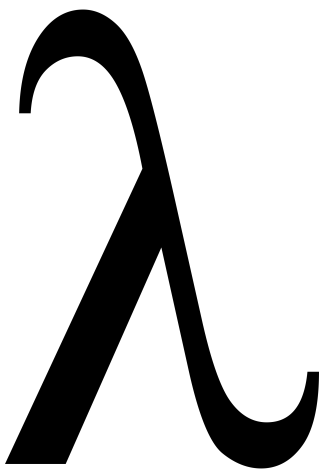
Lambda Expressions in Java

Erhan Bagdemir

bagdemir.com - Follow on @ebagdemir

February 9, 2015

Today's Agenda



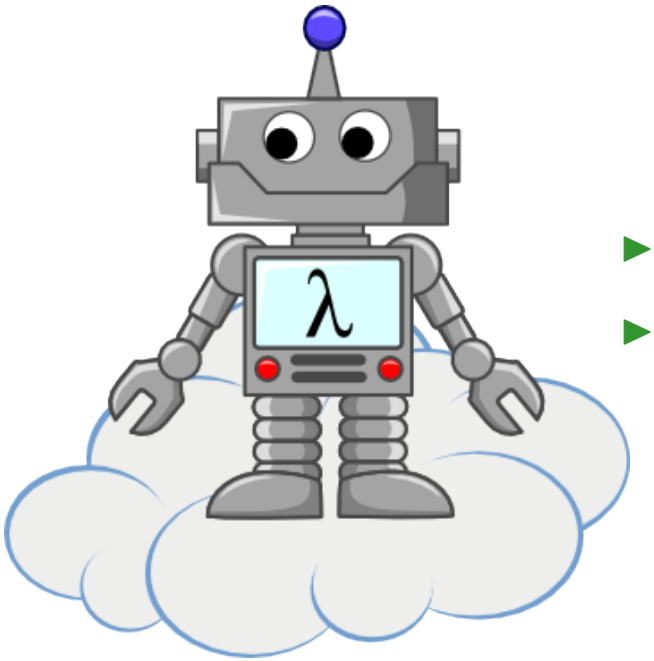
- Definition of Lambda.
- Lambdas in Java.
- Lambdas in ByteCode.
- Lambda examples.
  - forEach(...) and default implementations
  - Streams
  - Parallel Streams.

Lambda Expressions: Definition of Lambda

- A formal system for expression the computational behaviour.
- Invented by Alonzo Church in 1930.
- Lambda expressions consist of many parentheses i.e in Y-combinator:

$$Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

Lambda Expressions: The Idea, behind



- Functions are first-class citizens.
- Lambda expressions are high order functions:
  - They take other functions as parameter.
  - They may return functions.

Lambda Expressions: Java 8

Some examples of function Definitions:

- **f(x) = x + 1**
- LISP: **(lambda (x) x + 1)**
- C++11: **[](int x) { return x + 1; }**
- Scala: **x => x + 1 or just _ + 1**
- Java 8: **(int x) -> x + 1**

Lambda Expressions: Type of Lambda Expressions

The type of the lambda expressions are defined in **java.util.function**

```
Function<Integer, String> toStr = (x) -> Integer.parseValue(x);
```

You can pass lambdas as parameter:

```
final Function<Double, Double> withVAT = (x : Double) -> { return
    x * 1.19; }
fulfilShipment(move, 50.0d);
//...
private static Double fulfilShipment(Function<Double, Double>
    priceWithVAT, Double netPrice)
{
    perform(priceWithVAT.apply(netPrice));
}
```

Lambda Expressions: Type of Lambda Expressions

If there is no functional interface in **java.util.function** package for your usage, create your own:

```
@FunctionalInterface
public interface WorkflowLambda {
    IAsset for(String clientId, String userId, MetadataView view);
}

WorkflowLambda workflow = (clientId, userId, view) ->
     executeWith(clientId, userId, view);
```

- Call *Functional Interfaces* or *Single Abstract Method* interfaces.

Lambda Expressions: Scope of Lambda Expressions

Scope of the lambda expressions, the scope of the enclosing type:

```
public class OrderManagement {

   private double shippingCost = 4.90d;

   private void calculateOrder() {
     final double vatRate = 1.19d;
     final Function<Double, Double> withVAT = (Double x) -> {
         return (x * vatRate) + this.shippingCost; };
   }
}
```

The local variables can be referenced by lambda expression as long as they are **effective final**.

Lambda Expressions: Method References

We can use the methods of existing classes as lambda expressions. Method references are syntactic shortcut to the lambda expressions:

```
// static <T> void sort(T[] a, Comparator<? super T> c)

Collections.sort(myList, (firstInt, secondInt) -> firstInt -
    secondInt);
Collections.sort(myList, FooComparator::compare);

// if we'd an existing comparator
class FooComparator {
   public static int compare(final Integer firstInt, final Integer
       secondInt) {
        return firstInt - secondInt;
   }
}
```

Lambda Examples: forEach()

- Collections know how to iterate through their elements.
- It's a better style than imperative external loops, which provides functional polymorphism.

```
final List<Integer> myList = new ArrayList<>(3);
myList.add(1);
myList.add(5);
myList.add(2);

myList.forEach(System.out::println);
```

An new method in "Collection" interface without breaking code ?

Lambda Examples: Default Implementations

Interfaces may have **default** implementations:

```
public interface Dog {

    public void wagTail();

    public default void bark() {
        System.out.println("Bark!");
    }
}
```

- The interfaces may contain static methods as well.

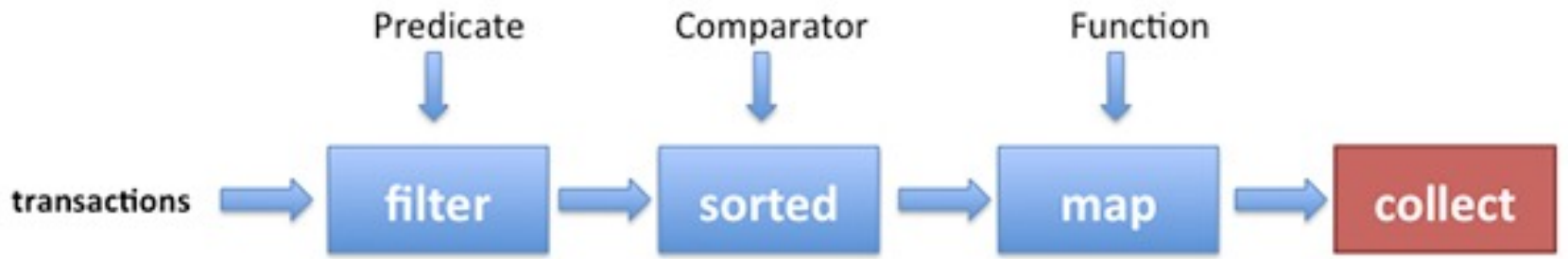
Streams

Lambda Examples: Streams

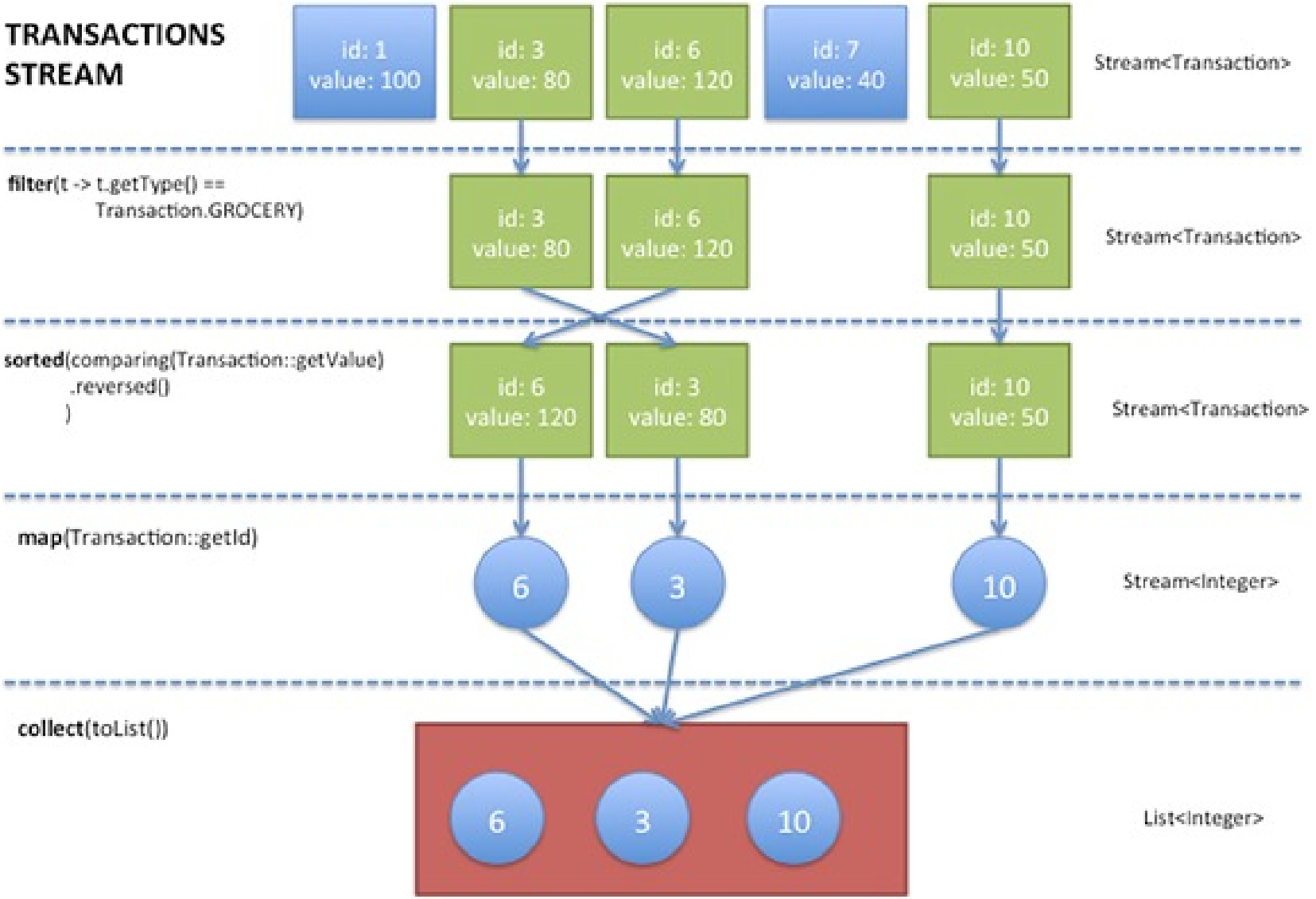
- Streams represent a sequence of elements which supports different kind of operations to perform computations on these elements.
- There are two types of operations:
  - Intermediate Operations (filter, sort, etc.).
  - Terminal Operations (collect, forEach, reduce, etc.)
- Stream features:
  - No storage.
  - Functional in nature.
  - Lazy in majority.
  - Allowed to be unbounded.
  - Consumable.
- Streams are monads!

Lambda Examples: Stream Pipelines

Processing pipeline consists of a source, intermediate operations and a terminal operation.



Lambda Examples: Stream Processing



Lambda Examples: Streams

- Stream sources: Arrays, Collections, Generators and I/O Streams, e.g:

```
try (BufferedReader br = new BufferedReader(new
     InputStreamReader(is))) {
    br.lines().forEach(System.out::println);
}
catch (Exception e) {
    System.out.println(e.getMessage());
}
```

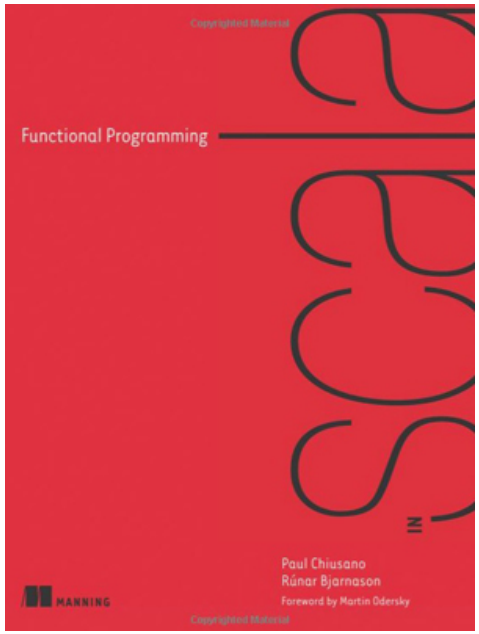
- After terminal operation, the stream is considered to be consumed.
- The intermediate operations are lazy and always return a new stream. (see Example 6)

Lambda Examples: Code Samples

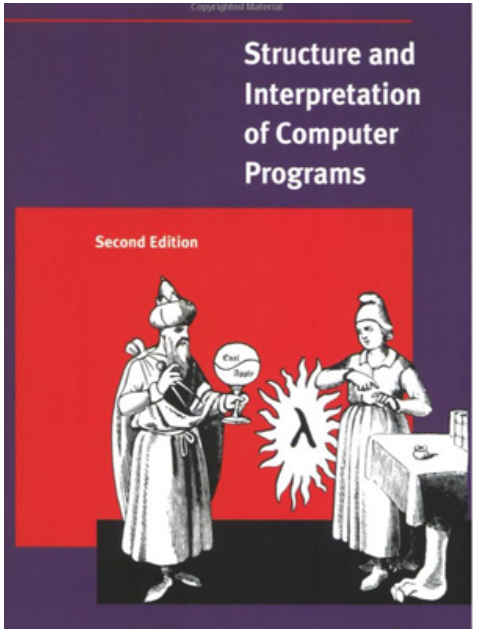
Some examples (https://github.com/bagdemir/java8-training) :

- Generators
- Comparators
- File I/O
- Regular Expressions
- Reducer
- Collectors
- Parallel Streams

Lambda Examples: Finish



Functional Programming (in Scala)



Structure and Interpretation of Computer Programs

Links

Lambda Quick Start

Oracle.com : http://bit.ly/1eR0we0

-

Streams

Oracle.com : http://bit.ly/1eh4aZo

Oracle.com : http://bit.ly/1ustKrM

-

Contact

https://github.com/bagdemir

https://twitter.com/ebagdemir

http://www.bagdemir.com